线性表

2019年2月2日 22:32

-
- 线性表

1. 线性结构的特点是数据元素之间的关系是线性的。数据元素可以看成是排列在一条线上或一个环上。线性表（Linear List）是最简单且最常用的一种数据结构
   1) 线性表是n(n≥0)个数据元素组成的有限序列。一般记作：
      i. L=(a1，a2，...，ai,... ,an)
   2) 例：有序数组归并
      i.
```
void MergeList(List La,List Lb,List Lc){
        InitList(Lc);
        i=j=1;k=0;
        La_len = ListLength(La);
        Lb_len = ListLength(Lb);
        while((i<=La_len)&&(j<=Lb_len)){       //La和Lb均非空
            GetElem(La , i , ai);
            GetElem(Lb , j , bj);
            if(ai<=bj) ListInsert(Lc,++k,ai);++i;
            else ListInsert(Lc,++k,bj);++j;  }
        //之后La和Lb某一项为空
        while(i <= La_len){
            GetElem(La , i++ , ai);
            ListInsert(Lc , ++k , ai); }
        while(j <= Lb_len){
            GetElem(Lb , j++ , bj);
            ListInsert(Lc , ++k , bj);  } }//MergeList
```

-
- 线性表的顺序表示

1. 顺序存储：用内存中一批地址连续的存储单元依次存储线性表中的数据元素
   1)
```
typedef  struct{
        ElemType elem[MAXSIZE];  //存储线性表中的元素
        int  len;      //线性表的当前表长
}SqList;
```
   2) 假设第一个元素存放的位置为b，每个元素占用的空间大小为L，则元素ai的存放位置为：LOC(ai) = b+L*(i-1)
   3) 插入：在线性表第i-1个数据元素和第i个数据元素之间插一个新的数据元素x
      i.
```
int Insert_Sq (SqList *L ,int i, ElemType x){
        if (i<1 || i>L->len+1) return 0;  /* 不合理的插入位置 i */
        if ( L->len == MAXSIZE-1) return -1;  /* 表已满 */
        for (j=L->len ; j>=i ; --j)
            L->elem[j+1] = L->elem[j];
            /* 插入位置及之后的元素右移*/
        L->elem[i] = x;   /*插入x */
        ++L->len;       /*表长加1 */
```

```
return 1; }
```

4) 删除：只需将ai+1，…，an依次向前移动一个位置，--长度

i.
```
int Delete_Sq (SqList *L ,int i )    {
 /* 删除线性表中第个i元素 */
    if (i<1 || i>L->len) return 0;   /*不合理的删除位置 i*/
    if (L->len==0) return -1;     /* 表已空*/
    for (j=i ; j<= L->len-1 ; j++)
        L->elem[j] = L->elem[j+1];
        /*被删元素后的元素前移 */
    --L->len; /*表长减1*/
    return 1;    }
```

ii. 插删顺序结构的平均操作次数都是n/2，时间复杂度均为O（n）

2. 动态顺序分配

1) 定义

i.
```
#define LIST_INIT_SIZE  100          //线性表存储空间的初始分配量
```

ii.
```
#define LISTINCREMENT  10          //线性表存储空间的分配增量
```

iii.
```
typedef struct {
    ElemType *elem;   //存储空间基址
    int length;          // 当前长度
    int listsize;          // 当前分配的存储容量
}SqList;
```

2) 初始化

i.
```
void InitList ( SqList & L ) {  // 构造一个空的线性表
    L.elem = ( ElemType * ) malloc( LIST_INIT_SIZE * sizeof ( ElemType ) );
    if ( !L.elem) exit("溢出！"); // 无空地址分配
        L.length = 0;
    L.listsize = LIST_INIT_SIZE;
    return OK;       }
```

3) 插入时判断空间不够

i.
```
if ( L.length>= L.listsize)    {
    newbase=(ElemType *) realloc(L.elem,
        (L.listsize+LISTINCREMENT) * sizeof(ElemType));
    if (!newbase) exit("溢出！");
    L.elem = newbase;         //新基址
    L.listsize+=LISTINCREMENT;  }  //增加存储容量
```

-
- 线性表的链式表示

1. 单链表

1) 定义：每个结点中只包含一个指针域，记录下一结点存储位置

i.
```
typedef  struct  Lnode{
    ElemType  data;      /*数据域*/
    struct Lnode  *next;  /* 指针域*/
}LNode;
```

2) 申请新结点：LNode *p=(LNode *) malloc (sizeof(LNode))（归还时free(p)）

3) 查找第i个结点（找不到时返回NULL）

i. LNode *Search(LNode *H, int i) {
/*H为表头结点的指针。存在时返回第i个结点 */
Lnode *p=H->next; /* P 指向首结点，j为计数器 */
int j=1; /*j为计数器 */
while (p && j++ < i) /* 查找第i个结点 */
p = p->next;
if (p== NULL || j > i)
return (NULL); /*不存在第i个结点 */
return p; } /*Search*/

4) 在P所指向的结点后插入x
i. void Insert_Linkst (LNode *H, LNode *p, ElemType x) {
s=(LNODE *)malloc(sizeof(LNODE)); /* 生成新结点 */
s->data=x;
s->next = p->next;
p->next=s; /* 修改指针域，完成插入 */

5) 在P所指向的结点前插入x
i. void Insert_Linkst (LNode *H, LNode *p, ElemType x) {
q=H;
while (q->next!=p)
q=q->next; /*寻找p结点的前趋 */
s=(LNode *) malloc (sizeof(LNode));
s->data=x;
s->next=p;
q->next=s;
} /* Insert_Linkst*/

6) 在值为x的结点前插入值为y的新结点（如果x值不存在，就插在表尾）
i. void Insertx_Linkst (LNode *H, ElemType x, ElemType y) {
s=(LNode *) malloc (sizeof(LNode));
s->data=y;
q=H;
p=H->next;
while (p && p->data!=x) { /*寻找值为x的结点*/
q=p;
p=p->next; }
s->next=p; /* 插入*/
q->next=s; } /* Insertx_Linkst * /

7) 删除第i个元素，返回删除成功与否
i. int Delete_Linkst (LNode *H, int i) {
q=H;
p=H->next; /* q始终指向p的前驱*/
j=1;
while (p && j<i) { /* 寻找第i个结点*/
q=p;
p=p->next;
++j; }
if (!p)
return 0; /* i大于表长*/
q->next=p->next; /* 删除结点*/
free(p);

```
        return 1; }  /* Delete_Linkst */
```

8) 删除所有值为x的结点，返回删除结点数

i.
```
int Deletex_Linkst ( LNode *H , ElemType x) {
    q=H;
    p=H->next;
    count=0;
    while( p ) { /* 遍历整个链表*/
        if( p->data==x ){ //找到
            q->next=p->next;
            free(p);
            ++count; }
        else{//未找到，往后挪一个
            q=p;
            p=p->next; } }
    return count; } /* Deletex_Linkst */
```

2. 静态链表

1) 定义：用一维数组描述的链表

i.
```
#define MaxSize = 1000;    //静态链表大小
```

ii.
```
typedef  int ElemType;
```

iii.
```
typedef struct node {         //静态链表结点
    ElemType data;
    int link;
}SNode;
```

iv.
```
typedef struct {              //静态链表
    SNode Nodes[MaxSize];
    int newptr;               //当前可分配空间首地址
}SLinkList;
```

2) 初始化

i.
```
void InitList ( SLinkList SL ) { //链表初始为空
    SL.Nodes[0].link = 1;
    SL.newptr = 1;   //当前可分配空间从 1 开始
                     //建立带表头结点SL.Nodes[0]的空链表
    for ( int i = 1; i < MaxSize-1; i++ )
        SL.Nodes[i].link = i+1;   //构成空闲链接表
        SL.Nodes[MaxSize-1].link = 0; //链表收尾 }
```

3) 查找给定值的结点

i.
```
int Find ( SLinkList SL, ElemType x ) {
    int p = SL.Nodes[0].link;  //指针 p 指向链表第一个结点
    while ( p != 0 )  //逐个查找有给定值的结点
        if ( SL.Nodes[p].data != x)
            p = SL.Nodes[p].link;
        else
            break;
    return p; }
```

4) 查找第i个结点

i.
```
int Locate ( SLinkList SL, int i ) {
    int j;
    if ( i < 0 )
```

```
        return -1;  //参数不合理
    if ( i == 0 )
        return 0;
    j = 0,   p = SL.Nodes[0].link;
    while ( p != 0 && j < i ) { //循环查找第 i 号结点
        p = SL.Nodes[p].link;
        j++; }
    return p; }
```

5) 插入新结点到第i个结点前

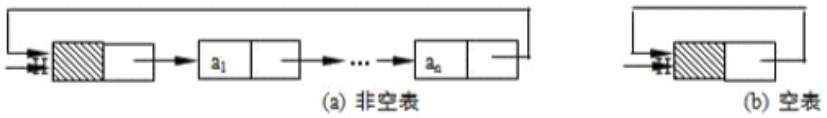
i.
```
int Insert ( SLinkList SL, int i, Elemtype x ) {
    int p,q;
    p = Locate ( SL, i-1 );  //在第i-1个点后插入，先定位
    if ( p == -1 )
        return 0;        //找不到结点
    q = SL. newptr;   //分配结点
    SL.newptr = SL.Nodes[SL.newptr].link;
    SL.Nodes[q].data = x;
    SL.Nodes[q].link = SL.Nodes[p].link;
    SL.Nodes[p].link = q;   //插入
    return 1; }
```

6) 释放第i个结点

i.
```
int Remove ( SLinkList SL, int i ) {
    int p, q;
    p = Locate ( SL, i-1 );
    if ( p == -1 )
        return 0;     //找不到结点
    q = SL.Nodes[p].link;   //第 i 号结点
    SL.Nodes[p].link = SL.Nodes[q].link;
    SL.Nodes[q].link = SL.newptr;     //把释放点链入空闲表中的头一个
    SL.newptr = q;
    return 1; }
```

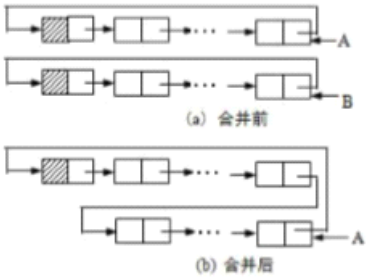
3. 循环链表：将单链表最后一个结点的指针域指向表头结点，形成一个环

1)



2) 将两个循环链表首尾相接（A、B是两链表尾结点指针）

i.



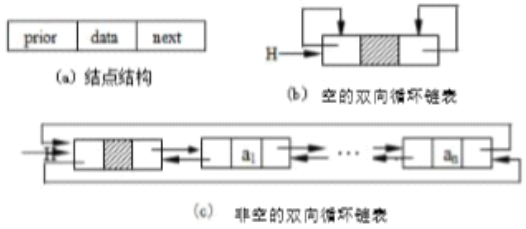
ii.
```
p=B->next;
B->next=A->next;
A->next=p->next;
free(p);
A=B; //好像没什么意义，可能是为了以A作为新表首
```

4. 双向链表

1) 定义：每个结点除了数据域外，还包含两个指针域，一个指向其后继结点，另一个指针指向其前趋结点

i. typedef struct dunode{
    ElemType data;
    struct dunode *prior, *next;
}DuNode;

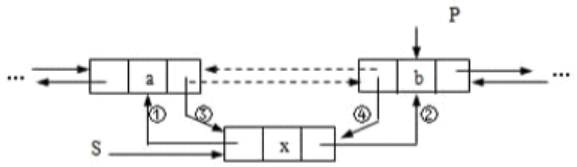
ii.



2) 插入新结点s在p前

i. s->prior=p->prior; /* ① */
s->next=p; /* ② */
p->prior->next=s; /* ③ */
p->prior=s; /* ④ */

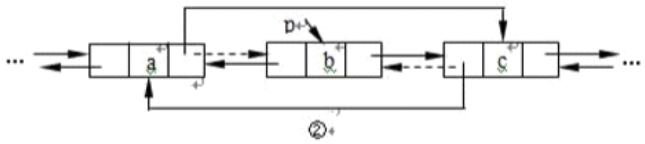
ii.



3) 删除结点p

i. p->prior->next=p->next; /* ① */
p->next->prior=p->prior; /* ② */
free(p);

ii.



-
- 一元多项式的表示及相加

1. 任务描述

在数学上，一个多项式可写成

$P(x)=a_nx^n+a_{n-1}x^{n-1}+\dots+a_1x+a_0$的形式，

1) 其中a_i为x_i的非零系数，例如

$A(x)=5x^9+8x^7+3x^2-12$。

2. 任务分析

一元多项式用线性表可表示为：

1) $P(x)=((a_n, n), (a_{n-1}, n-1), \dots (a_1, 1), (a_0, 0))$。

对于多项式$A(x)=5x^9+8x^7+3x^2-12$，则可表示为：

$A(x)=((5, 9), (8, 7), (3, 2), (-12, 0))$

2) 采用单链表来存储，则多项式中的每一项为单链表中的一个结点，每个结点包含三个域：系数域、指数域和指针域

3. 存储结构

1) typedef struct polynode {

```
    int coef; /*系数域*/
    int exp ; /*指数域*/
    struct polynode *next; /* 指针域*/
} PNode;
```

4. 算法
   1) 顺序输入n个元素的值，建立带表头结点的单链表
      i.
```
PNode *Creat_Linkst(int n) {
    PNode *head, *p, *s;
    int i;
    head = (PNode*)malloc(sizeof(PNode)); /* head为表头指针 */
    head->next = NULL;  /* 先建立一个带表头结点的空表*/
    p = head;   /*指针p始终指向当前链表的的尾结点*/
    printf("enter coef,exp:\n");
    for (i = 1; i <= n; ++i) {
        s = (PNode*)malloc(sizeof(PNode)); /* 生成新结点*/
        scanf("%d,%d", &s->coef, &s->exp);  /*输入系数和指数*/
        s->next = NULL;
        p->next = s;
        p = s;  /* 新结点插入在表尾*/ }
    return (head); }
```
   2) 输出
      i.
```
void Print_Linkst(PNode *H) {
    PNode *p; p = H->next;
    while (p->next) {
        if(p->coef!=0)
            printf("%dx^%d+", p->coef, p->exp); /*用x^表示xi */
        p = p->next; }
    if (p->exp)   /*尾结点的处理*/
        printf("%dx^%d\n", p->coef, p->exp);
    else
        printf("%d\n", p->coef);  /* 输出a0 */ }
```
   3) 相加
      i. 假设按照指数从大到小来表示一元多项式，相加的方法：一元多项式A(x)和B(x)相加后的结果覆盖A(x) 。将B(x)的所占内存释放
      ii. Pa和Pb为指向单链表A(x)和B(x)的当前结点指针。从链表的头开始比较各自的当前结点的指数，相同的将相加系数后改写Pa当前结点的系数项（若相加后系数为0则删除Pa指向的结点），Pa和Pb的当前结点指针分别后移
      iii. 若Pa的指数<Pb的指数, 则插入Pb的当前结点到Pa当前结点前，Pb的当前结点后移。若Pa的指数>Pb的指数,则把Pa的下一个结点作为当前结点
      iv. 继续ii
      v.
      vi.
      vii. --------我是底线----------